Introduction to VBE
Verified Boot for Embedded

An EFI-free boot standard

SPEAKER
Simon Glass

Agenda

- The problem
- Existing solutions and their drawbacks
- VBE boot flows
- VBE update flows
- VBE OS requests
- Future

Problem

- How should we architect the firmware and OS in an embedded system?
  - simple - no 1000-page specs
  - secure - only run signed software
  - updateable in the field
  - open so far as possible
  - unbrickable even with a bad update

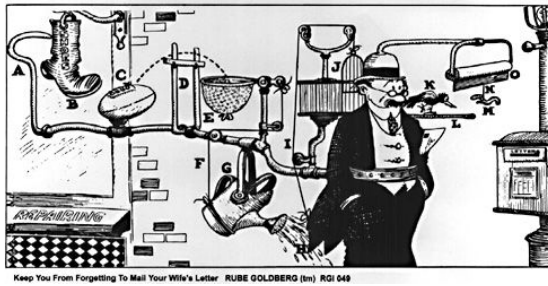
Secondary requirements

- Minimise read-only code, so updating can fix most bugs
- Firmware and OS components co-operate and communicate
- OS can request things from firmware, e.g. random numbers
- OS can control all updates

Existing solutions

- Tianocore / UEFI
  - Supports ~10 boards, effectively no shared source
- EFI support in U-Boot (EFI_LOADER)
  - Overly complex (see next slide)
- U-Boot with distroboot / scripts / Mender
  - Requires a fair bit of integration / effort
- U-Boot with standard boot
  - Same

Why not U-Boot's EFI_LOADER?



- Large and complex - spec is >2k pages
- Designed for closed source - UUIDs, dynamic libraries
- EFI_LOADER tries to avoid device tree, causing endless battles and workarounds
- Closely linked with ACPI, even if EFI_LOADER doesn't actually support it
- Around 100KB of code†
- Bolt-on implementation, although this is being addressed slowly
- Incompatible with existing U-Boot 'bootm', standard boot, FIT
- A closed-source trojan horse?

† rockpro64-rk3399 with EFI_RUNTIME_UPDATE_CAPSULE

Advantages of open firmware

- "Open Source where possible, with a shared code base"

- More peer review
- Better quality
- More secure
- Smaller
- Faster
- More widely used

In 10 years...

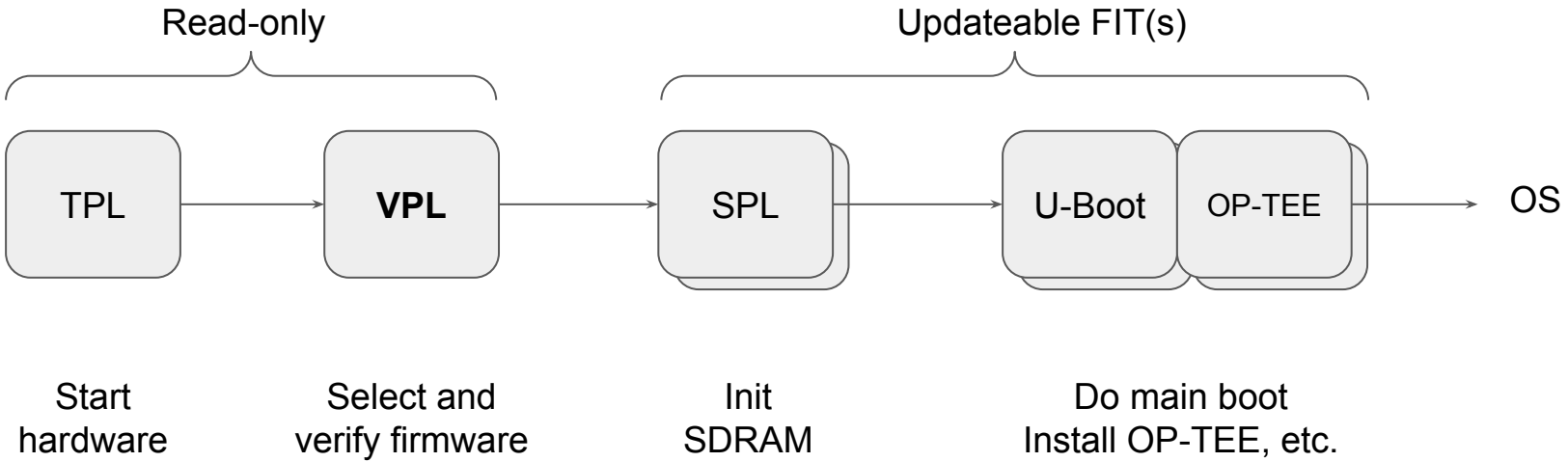
- What if we stick with EFI_LOADER?
  - Linux will only boot with its EFI Stub
  - Complexity will grow from the already-high base
  - Paves the way to end up at closed-source UEFI / private Tianocore trees
    - Tianocore has a permissive license almost no board support
    - It is **already** a failure in terms of open firmware
  - Security and flexibility will continue to suffer
- As an industry, we should promote open firmware

Verified Boot for Embedded - VBE

VBE Approach

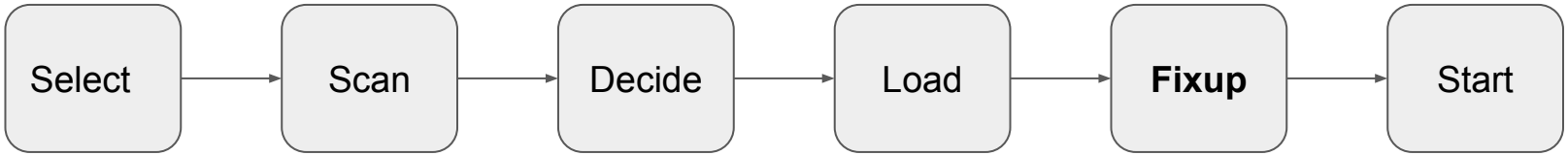
- Covers the whole lifecycle of firmware and OS
- Device tree is a first-class citizen
  - Used for configuration, communication to OS, etc.
- Relatively simple, full spec should be under 100 pages
- Builds on U-Boot's **industry-standard** features
  - FIT, including signature verification, compression, multiple images
  - Standard / distro boot, Driver model

Firmware boot flow



Firmware Handoff Protocol is used between phases

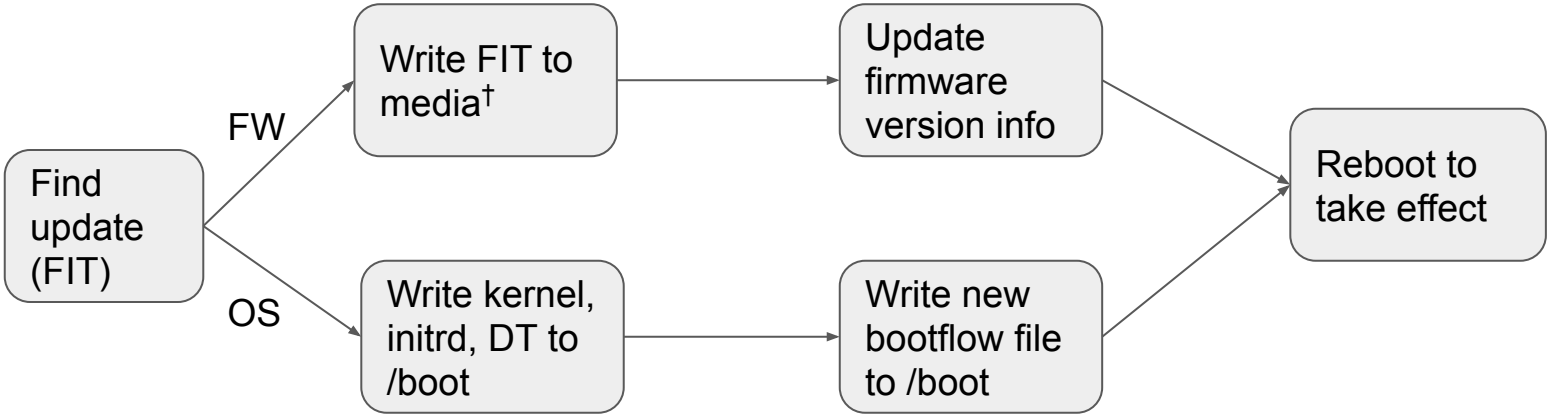
OS boot flow



Select	Scan	Decide	Load	**Fixup**	Start
Select boot devices	Find bootflows	Pick a bootflow	Load kernel, DT, ramdisk	Handle OS requests and add to OS device tree	Jump to the OS

This is essentially U-Boot's Standard Boot

Update flow (Linux with fwupd)



† May happen via a secure service

Firmware-update format

- Just a normal FIT with multiple configurations
  - One for things VPL loads (e.g. SPL)
  - One for things that SPL itself loads (e.g. U-Boot, OP-TEE)
- Version numbers are included
  - *vernum* is used for rollback protection

```
/ {
    description = "AP firmware";
    images { ‡
        spl {
            data-size = <0x000faf40>;†
            data-offset = <0x00000000>;
            description = "U-Boot SPL";
            type = "firmware";
            arch = "arm";
            os = "u-boot";
            compression = "none";
        };
        u-boot {
            data-size = <0x00ab0280>;
            data-offset = <0x000faf40>;
            description = "U-Boot";
            ...
    };
    configurations {
        conf-1-spl { ‡
            description = "AP Firmware SPL v1"
            version = "goat-r112";
            vernum = <0x00010003>;
            phase = "spl";
            firmware = "spl";
        };
        conf-1 {
            description = "AP Firmware v1";
            version = "goat-r112";
            vernum = <0x00010003>;
            firmware = "u-boot", "tee";
        };
    };
};
```

† Italics indicates fields generated by mkimage
‡ Signature / hashing fields are not shown

Kernel update format

- FIT containing kernel, DT, optional ramdisk
- Can also support separate files / packages
  - In that case the FIT metadata needs to be in its own package
  - e.g. package 'vbe-5.15.0-46' installs vbe-5.15.0-46.fit
- Supports multiple configurations
  - Can share kernels, but have one DT

```
/ {
    description = "Example kernel update";

    images {
    kernel-1 {
        data = /incbin/("vmlinuz-5.15.0");
        type = "kernel_noload";
            arch = "arm";
            os = "linux";
            compression = "gzip";
            random {  // OS requests random numbers
                compatible = "vbe,random-rand";
                vbe,size = <0x40>;
                vbe,required;
            };
        };
        fdt-1 {
            description = "ROCKPro64";
            data = /incbin/("rk3399-rockpro64.dtb");
            type = "flat_dt";
            arch = "sandbox";
            load = <%(fdt_addr)#x>;
            compression = "%(compression)s";
        };
    };
    configurations {
        default = "conf-1";
        conf-1 {
            compatible = "pine64,rockpro64-v2.1";
            kernel = "kernel-1";
            fdt = "fdt-1";
            ramdisk = ...
        };
    };
};
```

Scanning for bootflows

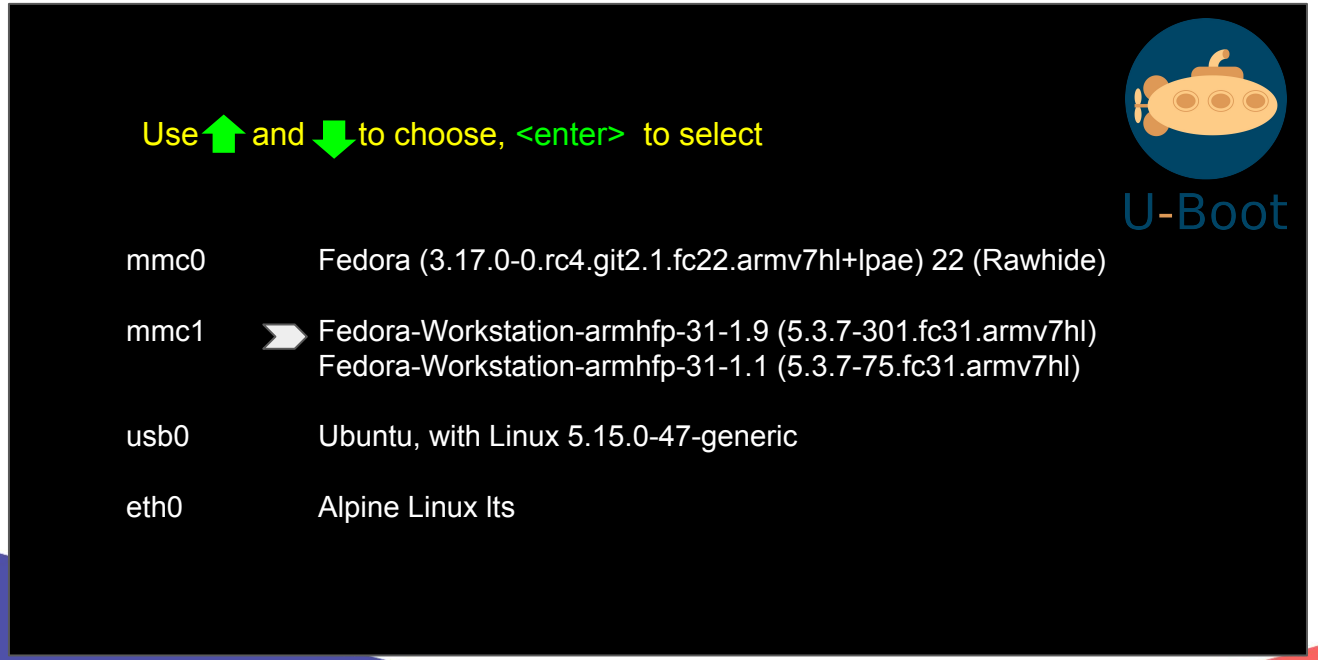
bootdevs
mmc0
mmc1
usb0
eth0

Use the VBE bootmeth to scan each for an OS

Lazy algorithm can stop when it finds something to boot

seq	bootdev	part	name
0	mmc0	5	Fedora (3.17.0-0.rc4.git2.1.fc22.armv7hl+lpae) 22 (Rawhide)
1	mmc1	3	Fedora-Workstation-armhfp-31-1.9 (5.3.7-301.fc31.armv7hl)
2	mmc1	3	Fedora-Workstation-armhfp-31-1.1 (5.3.7-75.fc31.armv7hl)
3	usb0	2	Ubuntu, with Linux 5.15.0-47-generic
4	eth0	0	Alpine Linux lts

Bootflow menu



VBE OS Requests

OS Requests

- OS requests are declared in the FIT
- U-Boot reads and processes each one
- Results are placed in the /chosen node
- Linux can read these as needed

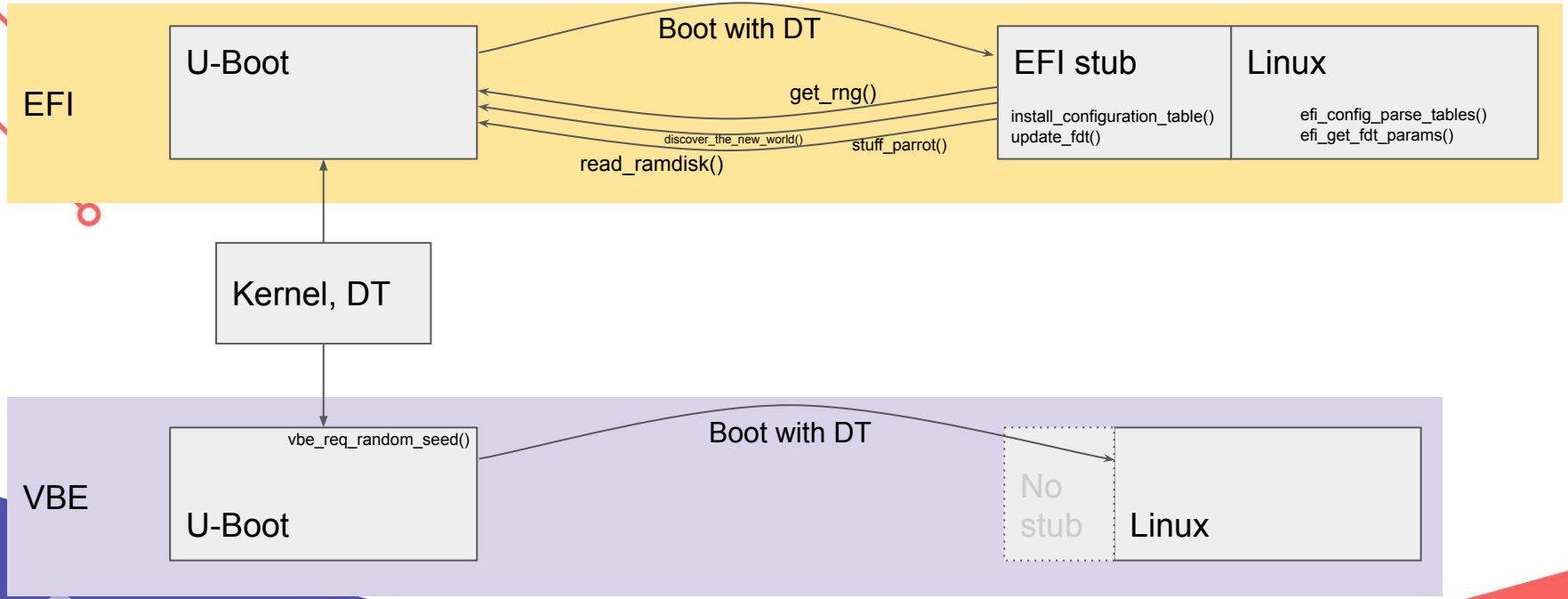
Input in FIT

```
images {
    kernel-1 {
        data = /incbin/("kernel.bin");
        type = "kernel_noload";
        os = "linux";

        random {
            compatible = "vbe,random-rand";
            vbe,size = <0x40>;
            vbe,required;
        };
    ...
```

Output in FDT passed to OS

```
chosen {
    bootargs = "console=tty0";

    random {
        compatible = "vbe,random-rand";
        vbe,size = <0x40>;
        vbe,required;
        data = [f7 bb 73 91 03 b3 14 17
                b1 14 2f cf 9f 8e 1d f5
                ...
                ];
    };
...
```

OS Requests: VBE versus EFI



VBE requests compared to EFI boot services

- No callbacks (simpler)
- Avoids double handling by U-Boot and EFI stub
- No need for the EFI stub
- Deterministic process
  - OS declares in advance what it needs
- A lot less code
- Easier to test

Linux 6.0 request types

Linux function name	LINUX_EFI_ guid_GUID or location	Purpose
check_platform_features()	arm_cpu_state_table	Install table of data with CPU info (arm32 only)
efi_convert_cmdline()	image->load_options	Convert cmdline from unicode u16 in to ASCII
efi_parse_options()	converted cmdline	nokaslr, quiet, noinitrd efi: nochunk, novamap, nosoftreserve, disable_early_pci_dma, no_disable_early_pci_dma, debug video: efifb=[...
setup_graphics() / efi_setup_gop()	uses graphics options from cmdline	Set up graphics
alloc_screen_info()	arm_screen_info_table	arm32 only
handle_kernel_image()		Select ASLR address (needs 4 random bytes)
efi_retrieve_tpm2_eventlog()	tpm_event_log tpm_final_log	Obtain TPM event log and final events
efi_enable_reset_attack_mitigation()	memory_only_reset_control (var)	Tell firmware to clear RAM on next reboot
efi_load_dtb() / handle_cmdline_files()		Load DTB from filesystem containing the kernel image
get_fdt()	device_tree	Load DTB from EFI config table
efi_load_initrd() / efi_load_initrd_dev_path() / efi_load_initrd_cmdline()	initrd_media	Load ramdisk from EFI, or failing that, initrd= cmdline args
efi_random_get_seed()	random_seed_table / rng_protocol	Get block of random data
efi_get_random_bytes()	rng_protocol	Select UEFI runtime services region (needs 4 random bytes)
install_memreserve_table()	memreserve_table	Get memory reservation table
allocate_new_fdt_and_exit_boot()		Write EFI, cmdline and initrd fields to the FDT. Exit boot services
efi_enter_kernel()		Enter the real kernel with fdt

Enabling VBE on a board

- Add a node for a VBE bootmeth
- This info appears in /chosen when booting the OS

U-Boot

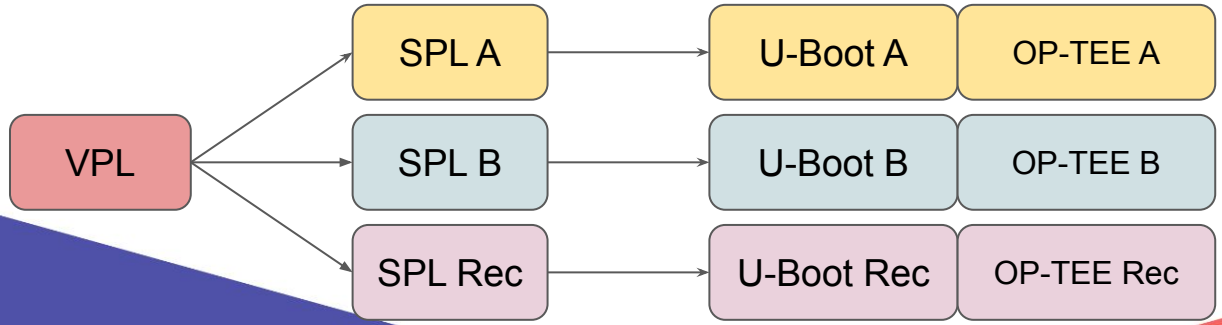
```
firmware0 {
      compatible = "fwupd,vbe-simple";
      storage = "mmc2";
      skip-offset = <0x200000>;
      area-start = <0>;
      area-size = <0xe00000>;
      state-offset = <0xe00000>;
      state-size = <0x40>;
      version-offset = <0xe00200>;
      version-size = <0x100>;
};
```

OS

```
chosen {
      compatible = "fwupd,vbe-simple";
      storage = "mmc2";
      ...
      cur-version = "goat-r112";
      cur-vernum = <0x0001003>;
      bootloader-version = "2022.10";
};
```

Beyond vbe-simple

- vbe-simple just supports a single firmware image
- Future work will add a vbe-abrec bootmeth
- This will support A and B images to cope with update failure
- Also a recovery image for when neither image boots
- Architecture is otherwise unchanged



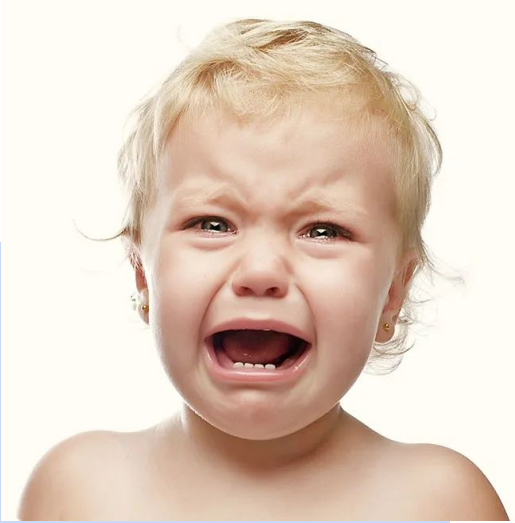
TPM and versions

- VBE uses a monotonically increasing *vernum* value
  - These map to user-readable version strings
  - TPM can be used to provide rollback protection
  - Update includes the minimum rollback version
- TPM can also provide remote attestation (e.g. for NetFlix)

Things related to VBE

- U-Boot standard boot https://u-boot.readthedocs.io/en/latest/develop/bootstd.html
- Firmware Handoff Protocol https://developer.arm.com/documentation/den0135/latest
- Distro boot at https://u-boot.readthedocs.io/en/latest/develop/distro.html
- fwupd plugin https://github.com/fwupd/fwupd/tree/main/plugins/vbe
- VBE docs are linked from https://u-boot.readthedocs.io/en/latest/develop/vbe.html
    - Except VBE OS Requests here

I'd love to use VBE, but...



- Objection 1: It is not an industry standard ("is not UEFI")
  - Well U-Boot is, at least on ARM / RISC-V
- Objection 2: VBE does not have a specification
  - It has 5 documents that will be become one in 2023
- Objection 3: We cannot ship the kernel and device tree in a FIT package
  - VBE does not require this
- Objection 4: VBE does not support ACPI
  - U-Boot supports ACPI on x86 and ARM (an OS Request!)
- Objection 5: VBE cannot handle grub customisation?
  - Some ideas exist, but for now just run grub if you have to

VBE status

- Major design is complete, but VBE is still in development
- U-Boot and fwupd support vbe-simple
- Initial U-Boot series for OS requests has been posted
- U-Boot VPL flow is under development (patches expected in November)
- Linux may need some code for OS requests, but perhaps not
- U-Boot boot menu is missing and grub needs work
- End-to-end demo with RockPRO64 is in progress (expected in January)
    - Will illustrate how to use VBE with and without grub
    - Full implementation and initial specifications expected in March 2023

How to get involved

- Read the documentation
- Review and contribute to the design
- Contribute code and ideas
- Try it out on your board

- Just say "**no**" to (U)EFI

Thank you - Questions?

Simon Glass
sjg@chromium.org
cc: u-boot@lists.denx.de

irc: sjg1

Backup material

Boot device (bootdev)

- The bootdev uclass provides sources for operating systems
  - e.g. a bootdev for MMC can access filesystems on the MMC device
  - e.g. a bootdev for Ethernet can tftp files
- May involve scanning for devices (e.g. USB)
- Ordering can be controlled
  - By default, fastest bootdevs are used first
- Boot devices are probed in a 'lazy' manner, for speed

```
mmc           0   [ + ]   bcm2835-sdhost        |   |-- mmc@7e202000
blk           0   [ + ]   mmc_blk               |   |   |-- mmc@7e202000.blk
bootdev       0   [   ]   mmc_bootdev           |   |   `-- mmc@7e202000.bootdev
mmc           1   [ + ]   sdhci-bcm2835         |   |-- sdhci@7e300000
blk           1   [   ]   mmc_blk               |   |   |-- sdhci@7e300000.blk
bootdev       1   [   ]   mmc_bootdev           |   |   `-- sdhci@7e300000.bootdev
```

Boot method (bootmeth)

- Uclass which knows how to find and load bootflows
  - e.g. distro boot can find an extlinux.conf file on a filesystem
  - e.g. Chromium OS bootmeth can locate Chrome OS on a bootdev
  - e.g. pxe can use this protocol to obtain the OS from the network
- Bootmeth can be configured with board-specific settings

```
bootmeths {
	compatible = "simple-bus";

	syslinux {
		compatible = "u-boot,distro-syslinux";
	};

	efi {
		compatible = "u-boot,distro-efi";
	};
};
```

Bootflow

- Not a device, just a data structure
- Bootflows are generated by:
  - Scanning bootdevs one by one
  - Using the available bootmeths on each bootdev

```
/**
 * struct bootflow_iter - state for iterating through bootflows
 *
 * This starts at with the first bootdev/partition/bootmeth and can b
 * iterate through all of them.
 *
 * Iteration starts with the bootdev. The first partition (0, i.e. wh
 * is scanned first. For partition 0, it iterates through all the ava
 * bootmeths to see which one(s) can provide a bootflow. Then it move
 * parition 1 (if there is one) and the process continues. Once all p
 * are examined, it moves to the next bootdev.
 *
 * Initially @max_part is 0, meaning that only the whole device (@par
 * used. During scanning, if a partition table is found, then @max_pa
 * updated to a larger value, no less than the number of available pa
 * This ensures that iteration works through all partitions on the bo
 *
 * @flags: Flags to use (see enum bootflow_flags_t)
 * @dev: Current bootdev
 * @part: Current partition number (0 for whole device)
 * @method: Current bootmeth
 * @max_part: Maximum hardware partition number in @dev, 0 if there i
 *      partition table
 * @err: Error obtained from checking the last iteration. This is use
 *      forward (e.g. to skip the current partition because it is not
 *      -ESHUTDOWN: try next bootdev
 * @num_devs: Number of bootdevs in @dev_order
 * @cur_dev: Current bootdev number, an index into @dev_order[]
 * @dev_order: List of bootdevs to scan, in order of priority. The so
 *      with the first one on the list
 * @num_methods: Number of bootmeth devices in @method_order
 * @cur_method: Current method number, an index into @method_order
 * @method_order: List of bootmeth devices to use, in order
 */
struct bootflow_iter {
```

Abstract

VBE supports firmware update, OS selection and other features that traditionally rely on the UEFI standard. It is simpler, more deterministic and easier to extend and test.

VBE comprises an implementation in U-Boot (using the new standard boot), firmware update in fwupd. It is designed to support verified boot (where SDRAM-init code and most firmware can be updated).

VBE is designed with embedded systems in mind. Rather than ignoring the existing boot methods and trying to shoe-horn UEFI into embedded systems, VBE extends the existing methods, such as U-Boot's Flat Image Tree.

This talk introduces the concept and the capability. It discusses the trade-offs that have been made ~~and ends with a demo on an ARM board~~.